CZ3006 Lab 4

Aim: Doing basic analysis of data log

In [ ]:
```
# Basic Libraries
import numpy as np
import pandas as pd
```

Creating the dataframe using pandas library

1. list the column names required
2. import the csv adding in the column names

In [ ]:
```
colName = ['Type', 'sflow_agent_address', 'inputPort', 'outputPort', 'src_Mac', 'dst_Mac', 'ethernet_type', 'in_vlan',
SFlow = pd.read_csv('./SFlow_Data_lab4.csv', header = None, names=colName, index_col=False)
SFlow = SFlow.loc[SFlow['Type'] == 'FLOW']
SFlow
```

EXERCISE 4A: TOP TALKERS AND LISTENERS

One of the most commonly used function in analyzing data log is finding out the IP address of the hosts that send out large amount of packet and hosts that receive large number of packets, usually know as TOP TALKERS and LISTENERS. Based on the IP address we can obtained the organization who owns the IP address.

The organizations are found using the link https://whatismyipaddress.com/

In [ ]:
```
#aim is to find the top 5 talkers using IP address
#table1 = SFlow.loc[SFlow['Type'] == 'FLOW']
table1 = SFlow
table1 = pd.DataFrame(table1['src_ip'])
Top5Talkers = table1.value_counts() #by default the resulting object will be in descending order
Top5Talkers.head(5)
```

In [ ]:
```
#aim is to find the top 5 listeners using IP address
#table1 = SFlow.loc[SFlow['Type'] == 'FLOW']
table1 = SFlow
table1 = pd.DataFrame(table1['dst_ip'])
Top5Listeners= table1.value_counts() #by default the resulting object will be in descending order
Top5Listeners.head(5)
```

EXERCISE 4B: TRANSPORT PROTOCOL

Using the IP protocol type attribute, determine the percentage of TCP and UDP protocol

In [ ]:
```
table1 = pd.DataFrame(SFlow['IP_protocol']) #take out IP_protocol only
IPprotocol = table1.value_counts() #count the number of unique occurrences, by default it is in descending order
IPprotocol = IPprotocol.to_frame().reset_index() #reset the index
IPprotocol.columns = ["IP_Protocol", "packets"] #rename the column
print(IPprotocol) #print the table
totalpackets = IPprotocol['packets'].sum() #sum the total number of packets
print()
# printing all the statistics
print("Total number of packets sent : ", totalpackets)
print("Percentage of UDP", 10*" ", ": ", np.array(IPprotocol.loc[IPprotocol['IP_Protocol'] == 17])[0][1]/totalpackets*
print("Percentage of TCP", 10*" ", ": ", np.array(IPprotocol.loc[IPprotocol['IP_Protocol'] == 6])[0][1]/totalpackets*1
```

EXERCISE 4C: APPLICATIONS PROTOCOL

Using the Destination IP port number determine the most frequently used application protocol. (For finding the service given the port number https://www.adminsub.net/tcp-udp-port-finder/)

In [ ]:
```
table1 = pd.DataFrame(SFlow['udp_dst_port/tcp_dst_port']) #table1 extracting from main SFlow
Top5Apps = table1.value_counts() #count the unique values e.g. 6 and 17 in descending
Top5Apps = Top5Apps.to_frame().reset_index() #reset the index to make it continuous
Top5Apps.columns = ["udp_dst_port/tcp_dst_port", "packets"] #renaming the columns
print(Top5Apps.head(5)) #printing the top 5
totalpackets = Top5Apps['packets'].sum() #sum the total number of packets sent by all application protocols
print()
print("Total number of packets sent : ", totalpackets)
```

EXERCISE 4D: TRAFFIC

The traffic intensity is an important parameter that a network engineer needs to monitor closely to determine if there is congestion. You would use the IP packet size to calculate the estimated total traffic over the monitored period of 15 seconds. (Assume the sampling rate is 1 in 2048)

IP packet size corresponds to the IP_Size column name

In [ ]:
```
table1 = pd.DataFrame(SFlow['IP_Size']) #table1 extracting from main SFlow
totalbytes = table1.sum()
totalMB = totalbytes* 2048 / 1024 /1024 #the sampling rate is one in 2048 hence multiply by 2048
# multiply by 10^6 to change from bytes to MB
print("Total Traffic (MB) : ", np.array(totalMB)[0]) #print result
```

EXERCISE 4E: ADDITIONAL ANALYSIS

Please append ONE page to provide additional analysis of the data and the insight it provides. Examples include: Top 5 communication pairs; Visualization of communications between different IP hosts; etc. Please limit your results within one page (and any additional results that fall beyond one page limit will not be assessed).

Finding the top 5 communication pair

We have to take into account the fact that src --> dst and dst --> src both have to be counted as a communication pair. For example, two such entries:

src | dst 137.132.228.15 193.62.192.8
193.62.192.8 137.132.228.15

means between the IP addresses `137.132.228.15` and `193.62.192.8` there have been **2** communications.

In [ ]:
```
#creating two tables with src_ip and dst_ip
table1 = pd.DataFrame(SFlow[['src_ip', 'dst_ip']])
table2 = pd.DataFrame(SFlow[['src_ip', 'dst_ip']])

#renaming the columns to reverse the IP address for the scenario explained in the markdown above
table1.rename(columns = {'dst_ip': 'IP1', 'src_ip': 'IP2'}, inplace = True)
table2.rename(columns = {'dst_ip': 'IP2', 'src_ip': 'IP1'}, inplace = True)

finaltable = pd.concat([table1, table2]) #concatenate the table

#group by IP1 AND IP2 together count them, sort them in descending and reset the index
commpair = (finaltable.groupby(['IP1', 'IP2'])).size().sort_values(ascending=False).reset_index(name='count')

#print(commpair) --> results appear here itself but to make it neater ---------------------------------------------

commpair['index'] = commpair.index #create another column for index to be used to remove swapped duplicates
#split commpair table by IP1 and IP2 keeping index and count
df_1 = commpair[['IP1', 'index', 'count']]
df_2 = commpair[['IP2', 'index', 'count']]

#rename such that they have the same column names
df_1.columns = ['IP', 'index', 'count']
df_2.columns = ['IP', 'index', 'count']

#Keep track of which table they originally came from
df_1['source'] = 1
df_2['source'] = 2

#concatenate them back
df = pd.concat([df_1, df_2])

#sort them by index dropping the duplicates
out = df.sort_values(['index']).drop_duplicates(['IP'], keep='first')

#extract them based on their source of origin
df_1_out = out[out['source'] == 1][['IP', 'count', 'index']]
df_2_out = out[out['source'] == 2][['IP', 'count', 'index']]

#finally merge them and drop the index column and count_1 column
final = df_1_out.merge(df_2_out, on='index', suffixes=('_1', '_2')).drop('index', axis=1)
final = final.drop(columns = ['count_1'])

#rename the count column
final.rename(columns = {'count_2': 'count'}, inplace=True)

#view the top 5 commmunication pairs
print(final.head(5))
```

Visualizing these communication pairs (By IP Address)

There are two parts to this:

1. Visualising based on the IP address of the sending and receiving hosts
2. Based on the location where these IP addresses are originating from

In [ ]:
```
commpair = final.head(500) #extract the top 500
'''
after index 466, it is only one time communication but for a more whole number and
```

```
to include some one time communication 500 was chosen
'''
```

In [ ]:
```
import sys
!{sys.executable} -m pip install pyvis
# Module used for the graph
```

In [ ]:
```
from pyvis.network import Network #import the necessary libraries and modules
```

In [ ]:
```
commgraph = Network(notebook = True)  # create an empty graph

#create two tables one for IP1 and one for IP2. This is because some IP can ONLY be present in IP2 or IP1
iphost = pd.DataFrame(commpair[['IP_1']])
iphost2 = pd.DataFrame(commpair[['IP_2']])

#rename the columns
iphost.rename(columns = {'IP_1': 'IP'}, inplace = True)
iphost2.rename(columns = {'IP_2': 'IP'}, inplace = True)

#concatenate the tables
finaltable = pd.concat([iphost, iphost2])

#drop duplicate IP addresses and reset the index
finaltable = finaltable.drop_duplicates('IP').reset_index(drop=True)

#print(finaltable)

for i in finaltable.index:
    commgraph.add_node(str(finaltable.loc[i, "IP"])) #adding nodes to the empty graph

#commgraph.show('nodes.html')
```

In [ ]:
```
# the list of edges is stored in commpair
for i in commpair.index:
    '''
    E.g. Index 0 of the communication pair
    IP1             |  IP2          | count
    137.132.228.15     193.62.192.8    4951
    between 137.132.228.15 and 193.62.192.8 we add an edge and we add a weight (called value) of 4951
    continue for the rest of the rows
    '''
    commgraph.add_edge(str(commpair.loc[i, "IP_1"]), str(commpair.loc[i, "IP_2"]), value = int(commpair.loc[i, "count'

commgraph.show_buttons(filter_=True) #for more interactive display of the graph
commgraph.show('nodes.html')
```

Visualizing these communication pairs (By Location)

There are two parts to this:

1. Visualising based on the IP address of the sending and receiving hosts
2. Based on the location where these IP addresses are originating from

In [ ]:
```
!{sys.executable} -m pip install IP2Location
import os
import IP2Location

#country binary file and documentation taken from https://www.ip2location.com/developers/python
database = IP2Location.IP2Location(os.path.join("IP-COUNTRY.BIN"))

#test with one ip address whether the code in documentation works
ip = '137.132.228.15'
rec = database.get_all(ip)
print(rec.country_long)
print(rec.country_short)
```

In [ ]:
```
locgraph = Network(notebook = True) #creating an empty graph
for i in finaltable.index:
    rec = database.get_all(str(finaltable.loc[i, "IP"])) #get location data of IP address
    #make sure IP address corresponds to the right country and it is available in the binary file
    if (rec.country_long != 'INVALID IP ADDRESS' and rec.country_long != 'IPV6 ADDRESS MISSING IN IPV4 BIN' and rec.co
        locgraph.add_node(str(rec.country_long)) #add the node to empty graph
```

In [ ]:
```
# the list of edges is stored in commpair
for i in commpair.index:
    rec1 = database.get_all(str(commpair.loc[i, "IP_1"])) #get country of first IP
    city1 = str(rec1.country_long) #convert to string
    rec2 = database.get_all(str(commpair.loc[i, "IP_2"])) #get country of second IP
    city2 = str(rec2.country_long) #convert to string

    if (city1 != 'INVALID IP ADDRESS' and city1 != 'IPV6 ADDRESS MISSING IN IPV4 BIN' and city1 != '-'):
        if (city2 != 'INVALID IP ADDRESS' and city2 != 'IPV6 ADDRESS MISSING IN IPV4 BIN' and city2 != '-'):
            #print(city1, city2)
            locgraph.add_edge(city1, city2, value = int(commpair.loc[i, "count"])) #add the edge to the graph


#show the graph
locgraph.repulsion(node_distance=70, spring_length=250)
locgraph.show_buttons(filter_=True)
locgraph.show('locgraph.html')
```

Visualization by ISP

For this part, only the top 200 communications were taken. Due to excessive requests to the API, it frequently resulted in timeout error. Also, in fact we only need the top 10 or 20 to see the ISPs that have the highest communication. This is because we are focused on looking at high network traffic in order to avoid congestion.

However, the graph is NOT included in the submitted word document due to constraints.

In [ ]:

```
import requests
# the list of edges is stored in commpair
ispgraph = Network(notebook = True)
for i in range(200):
    #request the isp of the ip address from http://ip-api.com
    rec1 = requests.get(str("http://ip-api.com/csv/"+ str(commpair.loc[i, "IP_1"]) + "?fields=isp"))
    rec2 = requests.get(str("http://ip-api.com/csv/"+ str(commpair.loc[i, "IP_2"]) + "?fields=isp"))
    ispgraph.add_node(str(rec1.text))  #add node of IP1
    ispgraph.add_node(str(rec2.text))  #add node of IP2
    ispgraph.add_edge(str(rec1.text), str(rec2.text), value = int(commpair.loc[i, "count"])) #Add edge between IP1 anc
```

In [ ]:

```
#print the graph
ispgraph.repulsion(node_distance=100, spring_length=200)
ispgraph.show_buttons(filter_=True)
ispgraph.show('ispgraph.html')
```

In [ ]: